CS1110 Spring 2016

Assignment 2: Due Friday Feb 26 at 6pm on CMS

Updates in orange; last update 9:15pm Tue Feb 16

You must work either on your own or with one partner. If you work with a partner, you and your partner must first register as a group in CMS (this requires an invitation issued by one of you on CMS and the other of you accepting it on CMS) and then submit your work as a group.

You may discuss background issues and general solution strategies with others, but the programs you submit must be the work of just you (and your partner). We assume that you are thoroughly familiar with the discussion of academic integrity that is on the course website. Any doubts that you have about "crossing the line" should be discussed with a member of the teaching staff before the deadline.

Topics. More practice with string slicing and Boolean computation. Implementing functions and procedures. The idea of one function calling another. Writing Application Scripts. Using graphics procedures. The assignment is based on Lectures 4 and 5 and Lab 3. You should download and play with the associated demo files.

Step Zero. Make sure you've done the additional step 4 in the Python installation instructions on the course webpage.

1 Siri Sez

Imagine driving in a city where at every intersection you have exactly four options:

- Continue Straight
- Turn Left
- Turn Right
- Make a U-Turn

Assume that the streets are laid out so that they are aligned with the four compass points. This means that you are either driving North (N), West (W), South (S), or East (E). We can encode a 6-intersection journey with a length-6 string that is made up of the characters `N`, `W`, `S`, and `E`. Here is an example:

`'NWNEEW'`

And here is a table that explains what happened at each intersection:

Intersection	The Relevant Substring	Action
0	`'N'`	Start Driving North
1	`'NW'`	Turn Left
2	`'WN'`	Turn Right
3	`'NE'`	Turn Right
4	`'EE'`	Continue Straight
5	`'EW'`	Make a U-Turn

In this problem you are to write Python code that takes a length-6 string like `'NWNEEW'` and prints out the six actions associated with the journey. Sample output:

```
NWNEEW
Start Driving North
Turn Left
Turn Right
Turn Right
Continue Straight
Make a U-Turn
```

We make some definitions in order to precisely describe the Python code that you are to write:

A length-6 string made up of the characters `N`, `W`, `S`, and `E` is called a *route string*. If `R` is a route string then `R[k]` indicates the direction of travel just after leaving the `k`th intersection. Thus, given the route string `R = 'NWNEEW'` we know that the driver leaves the second intersection heading north because the value of `R[2]` is `'N'`.

A length-2 string made up of the characters `N`, `W`, `S`, and `E` is called an *intersection string*. If `I` is an intersection string then `I[0]` encodes the direction of travel entering the intersection and `I[1]` encodes the direction of travel leaving the intersection. Thus, if `I = 'ES'` then the driver is driving east heading into an intersection and then turns south.

The strings `'Turn Right'`, `'Turn Left'`, `'Continue Straight'`, and `'Make a U-Turn'` are called *intersection instruction strings*.

The strings `'Start Driving North'`, `'Start Driving West'`, `'Start Driving South'`, and `'Start Driving East'` are called *starter strings*. They will be used to indicate what happens at the 0-th intersection, i.e., what happens at the start of the trip.

We are now ready to describe the functions and Application Script that you are to submit.

1.1 Getting Set Up

Using the editor, create a module `Siri.py`. The first version of this handout omitted the single quotes around '__main__' and 'Enter an Intersection String:'.

```
# Siri.py
# YOUR NAME(S) AND NETID(S) HERE
# DATE

""" Converts a route string into a sequence of driving instructions.
"""

def SiriSez(x):
    pass

def TripAdvisor(Route):
    pass

# Application Script
if __name__ == '__main__':
    I = raw_input('Enter an Intersection String: ')
    print I
```

Eventually this module will house two fully-implemented functions: `SiriSez` and `TripAdvisor`. Right now, the `pass` statements act as placeholders. If you run `Siri.py` it does something very simple. It prompts you to enter an intersection string like `'NW'` and then it prints it out. Try it!

1.2 Implementing `SiriSez`

Your next task is to implement `SiriSez(I)`. This function is to take an intersection string and return the appropriate intersection instruction string. Here are some examples:

Value of `I`	What `SiriSez` Returns
~~`'N'`~~	~~`'Start Driving North'`~~
`'EN'`	`'Turn Left'`
`'ES'`	`'Turn Right'`
`'EE'`	`'Continue Straight'`
`'EW'`	`'Make a U-Turn'`

Start your implementation of `SiriSez` *by removing the* `pass` *statement and writing the complete specification.* (Function specifications are discussed in Lecture 4.)

Developing the body of `SiriSez` requires a considerable amount of "Boolean thinking." There are lots of cases to consider but the amount of required code is modest if you take full advantage of the Boolean-related discussion in Lecture 3. As usual, there are several ways that the problem can be solved and it is fine if your approach uses a "helper function". Just make sure its implementation is part of your submitted `Siri.py`.

To debug your implementation of `SiriSez` you should augment the application script with

```
M = SiriSez(I)
print M
```

Thus, by running `Siri.py` you have a handy way of testing `SiriSez` on arbitrary input strings. Do not proceed until you are confident that your implementation of `SiriSez` is correct.

1.3 Implementing `TripAdvisor`

Next you are to implement `TripAdvisor`. This function takes a valid route string and prints seven lines of output:

Line 1 The route string.

Line 2. The associated starter string.

Lines 3-7. The associated intersection instruction strings.

Here is what `TripAdvisor('NWNEEW')` would print:

```
NWNEEW
Start Driving North
Turn Left
Turn Right
Turn Right
Continue Straight
Make a U-Turn
```

Note that `TripAdvisor` does not return a value. It is a *void* function. A void function is sometimes referred to as a procedure.

Start your implementation of `TripAdvisor` *by removing the* `pass` *statement and writing its specification.* In writing the body of `TripAdvisor`, you are required to make effective use of `SiriSez`. It is perfectly legal for one function (that you write) to call another function (that you write). Indeed, the point of having you develop `TripAdvisor` is to give you practice calling `SiriSez`.

~~To debug `TripAdvisor`~~ Modify the Application Script so that it solicits a valid route string and then calls `TripAdvisor`.

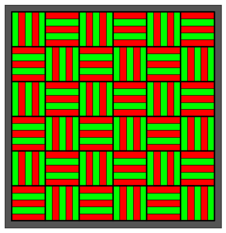
Submit `Siri.py` to CMS. It should include your finished implementations of `SiriSez`, `TripAdvisor`, and the Application Script just described. Finally, your implementations of `SiriSez` and `TripAdvisor` do not have to compensate for "bad input." Just worry about the correctness of your functions given that the preconditions are satisfied.

2 Parquet.com

In this problem you develop Python code that can produce more exciting Parquet designs than what you see in upscale flooring shops:

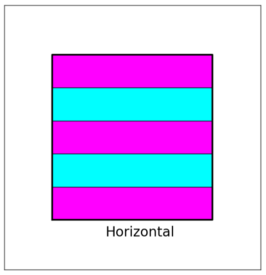
Lowes

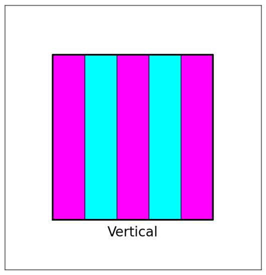
Home Depot



CS 1110

We start with the idea of a *Parquet tile.* A Parquet tile has two colors and an orientation:





Regardless of orientation, we assume that each tile is 5-by-5 and that it consists of five 1-by-5 strips that alternate in color. Strips 1, 3, and 5 have the same color and strips 2 and 4 have the same color. (Just to be clear, the strips in a horizontal tile are numbered bottom to top and the strips in a vertical tile are numbered left to right.) You will be using the `DrawRect` procedure from the `SimpleGraphics.py` module to draw each individual strip in the tile.

2.1 Getting Set Up

Download the modules `Parquet.py` and `SimpleGraphics.py` from the Assignments page on the course website. You will notice that `Parquet.py` includes two procedures (`DrawParquetTile` and `DrawParquetFloor`) that you are to implement and an Application Script to be used for testing. Since `Parquet.py` imports `SimpleGraphics.py` you must have both of these modules in the same folder/directory.

2.2 Implementing `DrawParquetTile`

Your first task is to implement a procedure that draws a Parquet tile with specified colors, orientation, and position:

```
def DrawParquetTile(a,b,H,C1,C2):
    """ Draws a 5-by-5 Parquet tile centered at (a,b).
    If H is True, then the strips are horizontal.
    If H is False, then the strips are vertical.
    Strips 1, 3, and 5 have color C1. Strips 2 and 4 have color C2.

    PreC: a and b are floats or ints. C1 and C2 are rgb lists,
    and H is Boolean.
    """
```

To make it look nice, use the default edgewidth and color when drawing the rectangular strips. After the strips are drawn put a black border around the whole 5x5 square. This can be done with the command

```
DrawRect(a,b,5,5,EdgeColor=BLACK,FillColor=None,EdgeWidth=3)
```

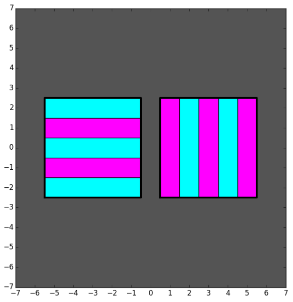
The given Application Script can be used to check out your implementation. All you have to do is run `Parquet.py`. The window labeled "Figure 1" should show two tiles drawn by your code. It might be behind two other windows labeled "Figure 2" and "Figure 3."

2.3 Implementing `DrawParquetFloor`

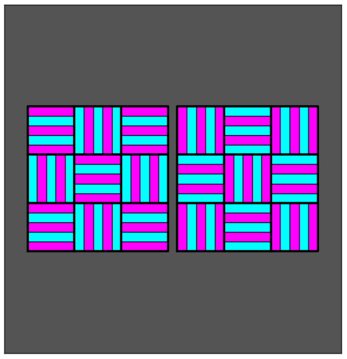
A *Parquet floor* is a 3-by-3 array of Parquet tiles. To precisely define such a layout we need a definition. Two Parquet tiles are *opposite* if

1. The first tile has the opposite orientation as the second tile.
2. Strips 1, 3, and 5 of the first tile have the same color as strips 2 and 4 of the second tile.
3. Strips 2 and 4 of the first tile have the same color as strips 1, 3, and 5 of the second tile.

Here are two Parquet tiles that are opposite:



In a Parquet floor, adjacent tiles are opposite. Here are two examples:



Notice that a Parquet floor is 15-strips-by-15-strips and that its center tile defines an orientation for the entire floor. In particular, we say that a Parquet floor has a horizontal (vertical) orientation if its center tile has a horizontal (vertical) orientation. In the above, the floor on the left is horizontal while the floor on the right is vertical. You are now ready to implement the following procedure:

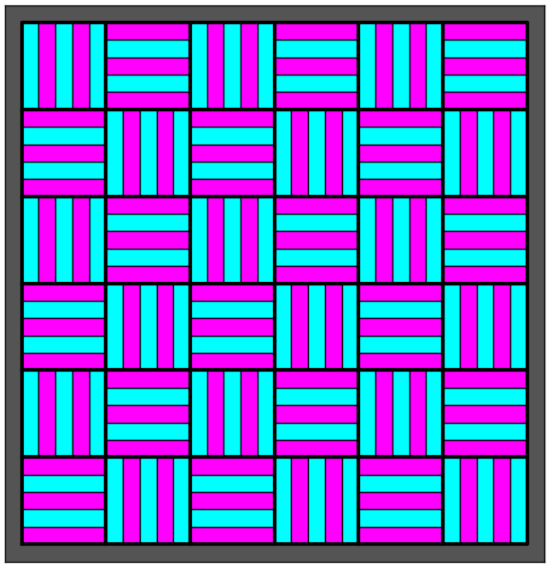
```
def DrawParquetFloor(a,b,H,C1,C2):
    """ Draws a 3x3 Parquet floor centered at (a,b).
    If H is True, then the center tile has horizontal strips.
    If H is False, then the center tile has vertical strips.
    Strips 1,3, and 5  of the center tile have color C1.
    Strips 2 and 4 of the center tile have color C2.
    PreC: a and b are floats or ints. C1 and C2 are rgb lists,
    and H is Boolean.
    """
```

Naturally, we are expecting your implementation to make calls to `DrawParquetTile`.

To check that things are working, ~~update~~ run the Application Script so that it displays both a horizontal and vertical Parquet floor in the second window. (The "Figure 2" window you want to check might be behind a window labeled "Figure 3.")

2.4 Dance Floor

Finally, display a *Dance Floor* by putting together four Parquet floors like this:



Adjacent tiles in the displayed dance floor must be opposite. Put the necessary code in the "third window" section of the Application Script.

Submit `Parquet.py` to CMS. It should include your finished implementations of `DrawParquetTile`, `DrawParquetFloor`, and the Application Script. *Make sure to include your name(s) and NetID(s) in a header comment.* And just to be clear, when `Parquet.py` is run, it should produce three windows. Window 1 displays two Parquet tiles with different orientations. Window 2 displays two Parquet floors with different orientations. And Window 3 displays the dance floor. You are free to use other colors besides `CYAN` and `MAGENTA`.

Post Script. Some of the codes you develop in this assignment involve lots of lines that almost "look alike." That's OK because right now we need practice calling functions. This means understanding the notion of an argument and the idea of passing the right value through an argument. Very soon we will learn about loops and iteration. With those ideas we will be able to rewrite very efficiently the "repetitive code" that you see in this assignment.